

Re: [RFC][PATCH] VFS: update documentation (take #2)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/6301.html>

From: Anton Altaparmakov (aia21_at_cam.ac.uk)

Date: 08/24/05

Date: Wed, 24 Aug 2005 20:56:08 +0100 (BST)

To: Pekka Enberg <penberg@cs.helsinki.fi>

Hi,

This is my second go at giving you my comments. The first time the email got lost due to a connection crash.)-: Anyway, its great to see this updated!

On Wed, 24 Aug 2005, Pekka Enberg wrote:

> *This patch brings the now out-of-date Documentation/filesystems/vfs.txt back to life. Thanks to Carsten Otte for the description on get_xip_page().*

>

> *Signed-off-by: Pekka Enberg <penberg@cs.helsinki.fi>*

> ----

>

> *vfs.txt | 382*

+++++-----

> *1 file changed, 291 insertions(+), 91 deletions(-)*

>

> *Index: 2.6-mm/Documentation/filesystems/vfs.txt*

>

=====

> *--- 2.6-mm.orig/Documentation/filesystems/vfs.txt*

> *+++ 2.6-mm/Documentation/filesystems/vfs.txt*

> *@@ -1,10 +1,9 @@*

> */* *- auto-fill *- */*

>

> *Overview of the Virtual File System*

>

> *- Richard Gooch <rgooch@atnf.csiro.au>*

> *+ Original author: Richard Gooch <rgooch@atnf.csiro.au>*

>

> *- 5-JUL-1999*

> *+ Last updated on August 24, 2005*

>

>

> *Conventions used in this document <section>*

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

> @@ -15,9 +14,6 @@ right-hand side of the section title. Ea
> "<subsection>" at the right-hand side. These strings are meant to make
> it easier to search through the document.
>
> -NOTE that the master copy of this document is available online at:
> -<http://www.atnf.csiro.au/~rgooch/linux/docs/vfs.txt>
> -
>
> What is it? <section>
> =====
> @@ -26,7 +22,7 @@ The Virtual File System (otherwise known
> Switch) is the software layer in the kernel that provides the
> filesystem interface to userspace programs. It also provides an
> abstraction within the kernel which allows different filesystem
> -implementations to co-exist.
> +implementations to coexist.
>
>
> A Quick Look At How It Works <section>
> @@ -77,7 +73,7 @@ back to userspace.
>
> Opening a file requires another operation: allocation of a file
> structure (this is the kernel-side implementation of file
> -descriptors). The freshly allocated file structure is initialised with
> +descriptors). The freshly allocated file structure is initialized with
> a pointer to the dentry and a set of file operation member functions.
> These are taken from the inode data. The open() file method is then
> called so the specific filesystem implementation can do it's work. You
> @@ -126,14 +122,18 @@ It's now time to look at things in more
> struct file_system_type <section>
> =====
>
> -This describes the filesystem. As of kernel 2.1.99, the following
> +This describes the filesystem. As of kernel 2.6.13, the following
> members are defined:
>
> struct file_system_type {
> const char *name;
> int fs_flags;
> - struct super_block *(*read_super) (struct super_block *, void *, int);
> - struct file_system_type * next;
> + struct super_block *(*get_sb) (struct file_system_type *, int,
> + const char *, void *);
> + void (*kill_sb) (struct super_block *);
> + struct module *owner;
> + struct file_system_type * next;
> + struct list_head fs_supers;
> };
>
> name: the name of the filesystem type, such as "ext2", "iso9660",
> @@ -141,51 +141,96 @@ struct file_system_type {

>
> *fs_flags*: various flags (i.e. *FS_REQUIRES_DEV*, *FS_NO_DCACHE*, etc.)
>
> - *read_super*: the method to call when a new instance of this
> + *get_sb*: the method to call when a new instance of this
> filesystem should be mounted
>
> - *next*: for internal VFS use: you should initialise this to *NULL*
> + *kill_sb*: the method to call when an instance of this filesystem
> + should be unmounted
> +
> + *owner*: for internal VFS use: you should initialize this to *NULL*
>
> -The *read_super()* method has the following arguments:
> + *next*: for internal VFS use: you should initialize this to *NULL*
> +
> +The *get_sb()* method has the following arguments:
>
> *struct super_block *sb*: the superblock structure. This is partially
> - initialised by the VFS and the rest must be initialised by the
> - *read_super()* method
> + initialized by the VFS and the rest must be initialized by the
> + *get_sb()* method
> +
> + *int flags*: mount flags
> +
> + *const char *dev_name*: the device name we are mounting.
>
> *void *data*: arbitrary mount options, usually comes as an ASCII
> string
>
> *int silent*: whether or not to be silent on error
>
> -The *read_super()* method must determine if the block device specified
> +The *get_sb()* method must determine if the block device specified
> in the superblock contains a filesystem of the type the method
> supports. On success the method returns the superblock pointer, on
> failure it returns *NULL*.
>
> The most interesting member of the superblock structure that the
> -*read_super()* method fills in is the "*s_op*" field. This is a pointer to
> +*get_sb()* method fills in is the "*s_op*" field. This is a pointer to
> a "*struct super_operations*" which describes the next level of the
> filesystem implementation.
>
> +Usually, a filesystem uses generic one of the generic *get_sb()*
> +implementations and provides a *fill_super()* method instead. The
> +generic methods are:
> +
> + *get_sb_bdev*: mount a filesystem residing on a block device
> +

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

> + *get_sb_nODEV*: mount a filesystem that is not backed by a device
> +
> + *get_sb_single*: mount a filesystem which shares the instance between
> + all mounts
> +
> + A *fill_super()* method implementation has the following arguments:
> +
> + *struct super_block *sb*: the superblock structure. The method *fill_super()*
> + must initialize this properly.
> +
> + *void *data*: arbitrary mount options, usually comes as an ASCII
> + string
> +
> + *int silent*: whether or not to be silent on error
> +
> +
> + *struct super_operations* <section>
> + =====
> +
> + This describes how the VFS can manipulate the superblock of your
> + *-*filesystem. As of kernel 2.1.99, the following members are defined:
> + *+*filesystem. As of kernel 2.6.13, the following members are defined:
> +
> + *struct super_operations* {
> + - *void (*read_inode) (struct inode *)*;
> + - *int (*write_inode) (struct inode *, int)*;
> + - *void (*put_inode) (struct inode *)*;
> + - *void (*drop_inode) (struct inode *)*;
> + - *void (*delete_inode) (struct inode *)*;
> + - *int (*notify_change) (struct dentry *, struct iattr *)*;
> + - *void (*put_super) (struct super_block *)*;
> + - *void (*write_super) (struct super_block *)*;
> + - *int (*statfs) (struct super_block *, struct statfs *, int)*;
> + - *int (*remount_fs) (struct super_block *, int *, char *)*;
> + - *void (*clear_inode) (struct inode *)*;
> + *struct inode *(*alloc_inode)(struct super_block *sb)*;
> + *void (*destroy_inode)(struct inode *)*;
> +
> + *void (*read_inode) (struct inode *)*;
> +
> + *void (*dirty_inode) (struct inode *)*;
> + *int (*write_inode) (struct inode *, int)*;
> + *void (*put_inode) (struct inode *)*;
> + *void (*drop_inode) (struct inode *)*;
> + *void (*delete_inode) (struct inode *)*;
> + *void (*put_super) (struct super_block *)*;
> + *void (*write_super) (struct super_block *)*;
> + *int (*sync_fs)(struct super_block *sb, int wait)*;
> + *void (*write_super_lockfs) (struct super_block *)*;
> + *void (*unlockfs) (struct super_block *)*;
> + *int (*statfs) (struct super_block *, struct kstatfs *)*;

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

```
> + int (*remount_fs) (struct super_block *, int *, char *);
> + void (*clear_inode) (struct inode *);
> + void (*umount_begin) (struct super_block *);
> +
> + void (*sync_inodes) (struct super_block *sb,
> + struct writeback_control *wbc);
> + int (*show_options)(struct seq_file *, struct vfsmount *);
> +
> + ssize_t (*quota_read)(struct super_block *, int, char *, size_t, loff_t);
> + ssize_t (*quota_write)(struct super_block *, int, const char *, size_t, loff_t);
> /;
>
> All methods are called without any locks being held, unless otherwise
> @@ -193,43 +238,58 @@ noted. This means that most methods can
> only called from a process context (i.e. not from an interrupt handler
> or bottom half).
>
> + alloc_inode: this method is called by inode_alloc() to allocate memory
> + for struct inode and initialize it.
> +
> + destroy_inode: this method is called by destroy_inode() to release
> + resources allocated for struct inode.
> +
> + read_inode: this method is called to read a specific inode from the
> - mounted filesystem. The "i_ino" member in the "struct inode"
> - will be initialised by the VFS to indicate which inode to
> - read. Other members are filled in by this method
> + mounted filesystem. The i_ino member in the struct inode is
> + initialized by the VFS to indicate which inode to read. Other
> + members are filled in by this method
```

May be worth mentioning that you can have read_inode set to NULL and use fs/inode.c::iget5_locked() instead of iget() to read inodes. This is necessary for filesystems for which the inode number is not sufficient to identify an inode.

```
> +
> + dirty_inode: this method is called by the VFS to mark an inode dirty.
>
> + write_inode: this method is called when the VFS needs to write an
> + inode to disc. The second parameter indicates whether the write
> + should be synchronous or not, not all filesystems check this flag.
>
> + put_inode: called when the VFS inode is removed from the inode
> - cache. This method is optional
> + cache.
>
> + drop_inode: called when the last access to the inode is dropped,
> + with the inode_lock spinlock held.
>
> - This method should be either NULL (normal unix filesystem
```

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

- > + *This method should be either NULL (normal UNIX filesystem semantics) or "generic_delete_inode" (for filesystems that do not want to cache inodes – causing "delete_inode" to always be called regardless of the value of i_nlink)*
- >
- > – *The "generic_delete_inode()" behaviour is equivalent to the*
- > + *The "generic_delete_inode()" behavior is equivalent to the*
- > *old practice of using "force_delete" in the put_inode() case,*
- > *but does not have the races that the "force_delete()" approach*
- > *had.*
- >
- > *delete_inode: called when the VFS wants to delete an inode*
- >
- > – *notify_change: called when VFS inode attributes are changed. If this*
- > – *is NULL the VFS falls back to the write_inode() method. This*
- > – *is called with the kernel lock held*
- > –
- > *put_super: called when the VFS wishes to free the superblock*
- > *(i.e. unmount). This is called with the superblock lock held*
- >
- > *write_super: called when the VFS superblock needs to be written to*
- > *disc. This method is optional*
- >
- > + *sync_fs: called when VFS is writing out all dirty data associated with*
- > + *a superblock. The second parameter indicates whether the method*
- > + *should wait until the write out has been completed. Optional.*
- > +
- > + *write_super_lockfs: called when VFS is locking a filesystem and forcing*
- > + *it into a consistent state. This function is currently used by the*
- > + *Logical Volume Manager (LVM).*
- > +
- > + *unlockfs: called when VFS is unlocking a filesystem and making it writable*
- > + *again.*
- > +
- > + *statfs: called when the VFS needs to get filesystem statistics. This*
- > + *is called with the kernel lock held*
- >
- > @@ –238,6 +298,17 @@ or bottom half).
- >
- > *clear_inode: called then the VFS clears the inode. Optional*
- >
- > + *umount_begin: called when the VFS is unmounting a filesystem.*
- > +
- > + *sync_inodes: called when the VFS is writing out dirty data associated with*
- > + *a superblock.*
- > +
- > + *show_options: called by the VFS to show mount options for /proc/<pid>/mounts.*
- > +
- > + *quota_read: called by the VFS to read from filesystem quota file.*
- > +
- > + *quota_write: called by the VFS to write to filesystem quota file.*

```

> +
> The read_inode() method is responsible for filling in the "i_op"
> field. This is a pointer to a "struct inode_operations" which
> describes the methods that can be performed on individual inodes.
> @@ -247,12 +318,11 @@ struct inode_operations
> =====
>
> This describes how the VFS can manipulate an inode in your
> -filesystem. As of kernel 2.1.99, the following members are defined:
> +filesystem. As of kernel 2.6.13, the following members are defined:
>
> struct inode_operations {
> - struct file_operations * default_file_ops;
> - int (*create) (struct inode *,struct dentry *,int);
> - int (*lookup) (struct inode *,struct dentry *);
> + int (*create) (struct inode *,struct dentry *,int, struct nameidata *);
> + struct dentry * (*lookup) (struct inode *,struct dentry *, struct nameidata *);
> int (*link) (struct dentry *,struct inode *,struct dentry *);
> int (*unlink) (struct inode *,struct dentry *);
> int (*symlink) (struct inode *,struct dentry *,const char *);
> @@ -261,25 +331,22 @@ struct inode_operations {
> int (*mknod) (struct inode *,struct dentry *,int,dev_t);
> int (*rename) (struct inode *, struct dentry *,
> struct inode *, struct dentry *);
> - int (*readlink) (struct dentry *, char *,int);
> - struct dentry * (*follow_link) (struct dentry *, struct dentry *);
> - int (*readpage) (struct file *, struct page *);
> - int (*writepage) (struct page *page, struct writeback_control *wbc);
> - int (*bmap) (struct inode *,int);
> + int (*readlink) (struct dentry *, char __user *,int);
> + int (*follow_link) (struct dentry *, struct nameidata *);
> + void (*put_link) (struct dentry *, struct nameidata *);

```

The symlink related operations just changed. It would be good to include the new syntax. See the recent thread on LKML and FSDEVEL titled "Kernel bug: Bad page state: related to generic symlink code and mmap". It is from last Friday and the weekend. (I started and Linus' posts contain the necessary info for you to fill in the gaps here.)

```

> void (*truncate) (struct inode *);
> - int (*permission) (struct inode *, int);
> - int (*smmap) (struct inode *,int);
> - int (*updatepage) (struct file *, struct page *, const char *,
> - unsigned long, unsigned int, int);
> - int (*revalidate) (struct dentry *);
> + int (*permission) (struct inode *, int, struct nameidata *);
> + int (*setattr) (struct dentry *, struct iattr *);
> + int (*getattr) (struct vfsmount *mnt, struct dentry *, struct kstat *);
> + int (*setxattr) (struct dentry *, const char *,const void *,size_t,int);
> + ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
> + ssize_t (*listxattr) (struct dentry *, char *, size_t);

```

```
> + int (*removexattr) (struct dentry *, const char *);
> };
>
> Again, all methods are called without any locks being held, unless
> otherwise noted.
>
> - default_file_ops: this is a pointer to a "struct file_operations"
> - which describes how to open and then manipulate open files
> -
> create: called by the open(2) and creat(2) system calls. Only
> required if you want to support regular files. The dentry you
> get should not have an inode (i.e. it should be a negative
> @@ -331,28 +398,135 @@ otherwise noted.
> inode it points to. Only required if you want to support
> symbolic links
>
> + put_link: called by the VFS to release resources allocated by
> + follow_link().
> +
> + truncate: called by the VFS to change the size of a file. The i_size
> + field of the inode is set to the desired size by the VFS before
> + this function is called. This function is called by the truncate(2)
> + system call and related functionality.
> +
> + permission: called by the VFS to check for access rights on a POSIX-like
> + filesystem.
> +
> + setattr: called by the VFS to set attributes for a file. This function is
> + called by chmod(2) and related system calls.
> +
> + getattr: called by the VFS to get attributes of a file. This function is
> + called by stat(2) and related system calls.
> +
> + setxattr: called by the VFS to set an extended attribute for a file.
> + Extended attribute is a name:value pair associated with an inode. This
> + function is called by setxattr(2) system call.
> +
> + getxattr: called by the VFS to retrieve the value of an extended attribute
> + name. This function is called by getxattr(2) function call.
> +
> + listxattr: called by the VFS to list all extended attributes for a given
> + file. This function is called by listxattr(2) system call.
> +
> + removexattr: called by the VFS to remove an extended attribute from a file.
> + This function is called by removexattr(2) system call.
> +
> +
> +struct address_space_operations <section>
> +=====
> +
> +This describes how the VFS can manipulate mapping of a file to page cache in
```

> +your filesystem. As of kernel 2.6.13, the following members are defined:
> +
> +struct address_space_operations {
> + int (*writepage)(struct page *page, struct writeback_control *wbc);
> + int (*readpage)(struct file *, struct page *);
> + int (*sync_page)(struct page *);
> + int (*writepages)(struct address_space *, struct writeback_control *);
> + int (*set_page_dirty)(struct page *page);
> + int (*readpages)(struct file *filp, struct address_space *mapping,
> + struct list_head *pages, unsigned nr_pages);
> + int (*prepare_write)(struct file *, struct page *, unsigned, unsigned);
> + int (*commit_write)(struct file *, struct page *, unsigned, unsigned);
> + sector_t (*bmap)(struct address_space *, sector_t);
> + int (*invalidatepage)(struct page *, unsigned long);
> + int (*releasepage)(struct page *, int);
> + ssize_t (*direct_IO)(int, struct kiocb *, const struct iovec *iov,
> + loff_t offset, unsigned long nr_segs);
> + struct page* (*get_xip_page)(struct address_space *, sector_t,
> + int);
> +};
> +
> + writepage: called by the VM write a dirty page to backing store.
> +
> + readpage: called by the VM to read a page from backing store.
> +
> + sync_page: called by the VM to notify the backing store to perform all
> + queued I/O operations for a page. I/O operations for other pages
> + associated with this address_space object may also be performed.
> +
> + writepages: called by the VM to write out all pages associated with the
> + address_space object.

Not "all" pages. I would simply remove the "all" and then it is fine,
just like you have for readpages below.

> +
> + set_page_dirty: called by the VM to set a page dirty.
> +
> + readpages: called by the VM to read pages associated with the address_space
> + object.
> +
> + prepare_write: called by the generic write path in VM to set up a write
> + request for a page.
> +
> + commit_write: called by the generic write path in VM to write page to
> + its backing store.
> +
> + bmap: called by the VFS to map a logical block offset within object to
> + physical block number. This method is use by for the legacy FIBMAP
> + ioctl. Other uses are discouraged.
> +

> + *invalidatepage*: called by the VM on truncate to disassociate a page from its
> + *address_space* mapping.
> +
> + *releasepage*: called by the VFS to release filesystem specific metadata from
> + a page.
> +
> + *direct_IO*: called by the VM for direct I/O writes and reads.
> +
> + *get_xip_page*: called by the VM to translate a block number to a page.
> + The page is valid until the corresponding filesystem is unmounted.
> + Filesystems that want to use *execute-in-place (XIP)* need to implement
> + it. An example implementation can be found in *fs/ext2/xip.c*.
> +
>
> *struct file_operations* <section>
> =====
>
> This describes how the VFS can manipulate an open file. As of kernel
> -2.1.99, the following members are defined:
> +2.6.13, the following members are defined:
>
> *struct file_operations* {
> *loff_t* (*llseek) (struct file *, *loff_t*, int);
> - *ssize_t* (*read) (struct file *, char *, *size_t*, *loff_t* *);
> - *ssize_t* (*write) (struct file *, const char *, *size_t*, *loff_t* *);
> + *ssize_t* (*read) (struct file *, char __user *, *size_t*, *loff_t* *);
> + *ssize_t* (*aio_read) (struct kiocb *, char __user *, *size_t*, *loff_t*);
> + *ssize_t* (*write) (struct file *, const char __user *, *size_t*, *loff_t* *);
> + *ssize_t* (*aio_write) (struct kiocb *, const char __user *, *size_t*, *loff_t*);
> *int* (*readdir) (struct file *, void *, *filldir_t*);
> *unsigned int* (*poll) (struct file *, *struct poll_table_struct* *);
> *int* (*ioctl) (struct inode *, struct file *, *unsigned int*, *unsigned long*);
> + *long* (*unlocked_ioctl) (struct file *, *unsigned int*, *unsigned long*);
> + *long* (*compat_ioctl) (struct file *, *unsigned int*, *unsigned long*);
> *int* (*mmap) (struct file *, *struct vm_area_struct* *);
> *int* (*open) (struct inode *, struct file *);
> + *int* (*flush) (struct file *);
> *int* (*release) (struct inode *, struct file *);
> - *int* (*fsync) (struct file *, *struct dentry* *);
> - *int* (*fasync) (struct file *, int);
> - *int* (*check_media_change) (*kdev_t* dev);
> - *int* (*revalidate) (*kdev_t* dev);
> + *int* (*fsync) (struct file *, *struct dentry* *, int *datasync*);
> + *int* (*aio_fsync) (struct kiocb *, int *datasync*);
> + *int* (*fasync) (int, struct file *, int);
> *int* (*lock) (struct file *, int, *struct file_lock* *);
> + *ssize_t* (*readv) (struct file *, const *struct iovec* *, *unsigned long*, *loff_t* *);
> + *ssize_t* (*writev) (struct file *, const *struct iovec* *, *unsigned long*, *loff_t* *);
> + *ssize_t* (*sendfile) (struct file *, *loff_t* *, *size_t*, *read_actor_t*, void *);
> + *ssize_t* (*sendpage) (struct file *, *struct page* *, int, *size_t*, *loff_t* *, int);
> + *unsigned long* (*get_unmapped_area)(struct file *, *unsigned long*, *unsigned long*, *unsigned long*,

unsigned long);

> + *int (*check_flags)(int);*

> + *int (*dir_notify)(struct file *filp, unsigned long arg);*

> + *int (*flock)(struct file *, int, struct file_lock *);*

> };

>

> *Again, all methods are called without any locks being held, unless*

> *@@ -362,8 +536,12 @@ otherwise noted.*

>

> *read: called by read(2) and related system calls*

>

> + *aio_read: called by io_submit(2) and other asynchronous I/O operations*

> +

> *write: called by write(2) and related system calls*

>

> + *aio_read: called by io_submit(2) and other asynchronous I/O operations*

aio_write, not aio_read, you did aio_read already...

> +

> *readdir: called when the VFS needs to read the directory contents*

>

> *poll: called by the VFS when a process wants to check if there is*

> *@@ -372,18 +550,25 @@ otherwise noted.*

>

> *ioctl: called by the ioctl(2) system call*

>

> + *unlocked_ioctl: called by the ioctl(2) system call. Filesystems that do not*

> + *require the BKL should use this method instead of the ioctl() above.*

> +

> + *compat_ioctl: called by the ioctl(2) system call when 32 bit system calls*

> + *are used on 64 bit kernels.*

> +

> *mmap: called by the mmap(2) system call*

>

> *open: called by the VFS when an inode should be opened. When the VFS*

> - *opens a file, it creates a new "struct file" and initialises*

> - *the "f_op" file operations member with the "default_file_ops"*

> - *field in the inode structure. It then calls the open method*

> - *for the newly allocated file structure. You might think that*

> - *the open method really belongs in "struct inode_operations",*

> - *and you may be right. I think it's done the way it is because*

> - *it makes filesystems simpler to implement. The open() method*

> - *is a good place to initialise the "private_data" member in the*

> - *file structure if you want to point to a device structure*

> + *opens a file, it creates a new "struct file". It then calls the*

> + *open method for the newly allocated file structure. You might*

> + *think that the open method really belongs in*

> + *"struct inode_operations", and you may be right. I think it's*

> + *done the way it is because it makes filesystems simpler to*

> + *implement. The open() method is a good place to initialize the*

> + *"private_data" member in the file structure if you want to point*
> + *to a device structure*
> +
> + *flush: called by the close(2) system call to flush a file*
>
> + *release: called when the last reference to an open file is closed*
>
> @@ -392,6 +577,23 @@ *otherwise noted.*
> + *fasync: called by the fcntl(2) system call when asynchronous*
> + *(non-blocking) mode is enabled for a file*
>
> + *lock: called by the fcntl(2) system call for F_GETLK, F_SETLK, and F_SETLKW*
> + *commands.*

<nitpick>

Remove the full stop at the end of the above. You haven't put full stops anywhere else...

</nitpick>

> +
> + *readv: called by the readv(2) system call*
> +
> + *writev: called by the readv(2) system call*
> +
> + *sendfile: called by the sendfile(2) system call*
> +
> + *get_unmapped_area: called by the mmap(2) system call*
> +
> + *check_flags: called by the fcntl(2) system call for F_SETFL command*
> +
> + *dir_notify: called by the fcntl(2) system call for F_NOTIFY command*
> +
> + *flock: called by the flock(2) system call*
> +
> + *Note that the file operations are implemented by the specific*
> + *filesystem in which the inode resides. When opening a device node*
> + *(character or block special) most filesystems will call special*
> + *@@ -400,9 +602,7 @@ driver information. These support routin*
> + *operations with those for the device driver, and then proceed to call*
> + *the new open() method for the file. This is how opening a device file*
> + *in the filesystem eventually ends up calling the device driver open()*
> + *-method. Note the devfs (the Device FileSystem) has a more direct path*
> + *-from device node to device driver (this is an unofficial kernel*
> + *-patch).*
> + *method.*
>
>
> + *Directory Entry Cache (dcache) <section>*
> + *@@ -415,14 +615,14 @@ This describes how a filesystem can over*
> + *operations. Dentries and the dcache are the domain of the VFS and the*
> + *individual filesystem implementations. Device drivers have no business*

> here. These methods may be set to NULL, as they are either optional or
> -the VFS uses a default. As of kernel 2.1.99, the following members are
> +the VFS uses a default. As of kernel 2.6.13, the following members are
> defined:
>
> struct dentry_operations {
> - int (*d_revalidate)(struct dentry *);
> + int (*d_revalidate)(struct dentry *, struct nameidata *);
> int (*d_hash) (struct dentry *, struct qstr *);
> int (*d_compare) (struct dentry *, struct qstr *, struct qstr *);
> - void (*d_delete)(struct dentry *);
> + int (*d_delete)(struct dentry *);
> void (*d_release)(struct dentry *);
> void (*d_iput)(struct dentry *, struct inode *);
> };
> @@ -471,7 +671,7 @@ manipulate dentries:
> "d_delete" method is called
>
> d_drop: this unhashes a dentry from its parents hash list. A
> - subsequent call to dput() will dellocate the dentry if its
> + subsequent call to dput() will deallocate the dentry if its
> usage count drops to 0
>
> d_delete: delete a dentry. If there are no other open references to
> @@ -507,16 +707,16 @@ up by walking the tree starting with the
> of the pathname and using that dentry along with the next
> component to look up the next level and so on. Since it
> is a frequent operation for workloads like multiuser
> -environments and webservers, it is important to optimize
> +environments and web servers, it is important to optimize
> this path.
>
> Prior to 2.5.10, dcache_lock was acquired in d_lookup and thus
> in every component during path look-up. Since 2.5.10 onwards,
> -fastwalk algorithm changed this by holding the dcache_lock
> +fast-walk algorithm changed this by holding the dcache_lock
> at the beginning and walking as many cached path component
> -dentries as possible. This significantly decreases the number
> +dentries as possible. This significantly decreases the number
> of acquisition of dcache_lock. However it also increases the
> -lock hold time significantly and affects performance in large
> +lock hold time significantly and affects performance in large
> SMP machines. Since 2.5.62 kernel, dcache has been using
> a new locking model that uses RCU to make dcache look-up
> lock-free.
> @@ -527,7 +727,7 @@ protected the hash chain, d_child, d_ali
> as d_inode and several other things like mount look-up. RCU-based
> changes affect only the way the hash chain is protected. For everything
> else the dcache_lock must be taken for both traversing as well as
> -updating. The hash chain updations too take the dcache_lock.
> +updating. The hash chain updates too take the dcache_lock.

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

- > *The significant change is the way d_lookup traverses the hash chain,*
- > *it doesn't acquire the dcache_lock for this and rely on RCU to*
- > *ensure that the dentry has not been *freed*.*
- > *@@ -540,9 +740,9 @@ very frequently occurring operations. Bo*
- > *of path names to find the dentry corresponding to the*
- > *concerned file. In 2.4 kernel, dcache_lock was held*
- > *during look-up of each path component. Contention and*
- > *-cacheline bouncing of this global lock caused significant*
- > *+cache-line bouncing of this global lock caused significant*
- > *scalability problems. With the introduction of RCU*
- > *-in linux kernel, this was worked around by making*
- > *+in Linux kernel, this was worked around by making*
- > *the look-up of path components during path walking lock-free.*
- >
- >
- > *@@ -562,7 +762,7 @@ Some of the important changes are :*
- > *2. Insertion of a dentry into the hash table is done using*
- > *hlist_add_head_rcu() which take care of ordering the writes -*
- > *the writes to the dentry must be visible before the dentry*
- > *- is inserted. This works in conjunction with hlist_for_each_rcu()*
- > *+ is inserted. This works in conjunction with hlist_for_each_rcu()*
- > *while walking the hash chain. The only requirement is that*
- > *all initialization to the dentry must be done before hlist_add_head_rcu()*
- > *since we don't have dcache_lock protection while traversing*
- > *@@ -584,7 +784,7 @@ Some of the important changes are :*
- > *the same. In some sense, dcache_rcu path walking looks like*
- > *the pre-2.5.10 version.*
- >
- > *-5. All dentry hash chain updations must take the dcache_lock as well as*
- > *+5. All dentry hash chain updates must take the dcache_lock as well as*
- > *the per-dentry lock in that order. dput() does this to ensure*
- > *that a dentry that has just been looked up in another CPU*
- > *doesn't get deleted before dget() can be done on it.*
- > *@@ -640,10 +840,10 @@ handled as described below :*
- > *Since we redo the d_parent check and compare name while holding*
- > *d_lock, lock-free look-up will not race against d_move().*
- >
- > *-4. There can be a theoretical race when a dentry keeps coming back*
- > *+4. There can be a theoretical race when a dentry keeps coming back*
- > *to original bucket due to double moves. Due to this look-up may*
- > *consider that it has never moved and can end up in a infinite loop.*
- > *- But this is not any worse than theoretical livelocks we already*
- > *+ But this is not any worse than theoretical livelocks we already*
- > *have in the kernel.*

Otherwise it looks great. We should make sure either Linus or Andrew apply it.

Best regards,

Anton

Re: [RFC][PATCH] VFS: update documentation (take #2)

Linux-Kernel: Re: [RFC][PATCH] VFS: update documentation (take #2)

--

Anton Altaparmakov <aia21 at cam.ac.uk> (replace at with @)
Unix Support, Computing Service, University of Cambridge, CB2 3QH, UK
Linux NTFS maintainer / IRC: #ntfs on irc.freenode.net
WWW: <http://linux-ntfs.sf.net/> & <http://www-stu.christs.cam.ac.uk/~aia21/>

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>